Amendments to the Claims

This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1. (Currently amended) A method for dynamic transformation of programs, said method operable at least in part within an information processing system, comprising:

~~a.~~ accessing a dynamic instrumentation interface of a mixed mode runtime environment, the accessing step comprising determining whether a first class is loaded; initiating an instrumentation client program; and making a predetermined transformation call to the interface; and

~~b.~~ controlling a compiler of the mixed mode runtime environment via the interface to transform a program, the controlling step comprising, if the first class is loaded,

controlling the compiler to compile the first class into an executable program and transform the executable program into a transformed program according to the predetermined transformation call; and, if the first class is not loaded, storing transformation information relating to the first class and

monitoring the mixed mode runtime environment to determine when the first class is loaded.

2. (Currently amended)The method of claim 1, wherein the mixed mode runtime environment is a virtual machine and the compiler is a just in time ~~(JIT)~~ compiler~~, step a further comprising determining whether a first class is loaded, and step b comprising, if the first class is loaded, controlling the JIT compiler to compile the first class into an executable program and transform the executable program into a transformed program~~.

3. (Currently amended)The method of claim 2, wherein the virtual machine is one of the group of a Java virtual machine and a .NET virtual machine~~, further comprising, if the first class is not~~

~~loaded, storing transformation information relating to the first class and monitoring the virtual machine to determine when the first class is loaded.~~

4. (Currently amended)The method of claim [[3]] 1, wherein the ~~steps~~ step of monitoring ~~and determining~~ the mixed mode runtime environment to determine when the first class is loaded ~~are~~ is performed by a class loader responsive to instructions via the interface.

5. (Currently amended)The method of claim [[2]] 1, ~~wherein the virtual machine is one of the group of a Java virtual machine and a .NET virtual machine,~~ further comprising, if the first class is loaded, allowing a current instantiation of the first class to run until terminated.

6. (Currently amended)The method of claim [[2]] 1, ~~wherein the virtual machine is one of the group of a Java virtual machine and a .NET virtual machine,~~ further comprising, when the first class is ~~already~~ loaded, replacing currently running code based on an old-object method of the first class with the transformed program and adjusting an activation stack so existing invocations of the old-object method continue executing after adjusting the activation stack.

7. (Currently amended)The method of claim 1, wherein ~~step a further comprises~~

~~initiating an instrumentation client program, and~~

~~the step of accessing comprises making a predetermined transformation call to the interface, and step b.~~ the controlling step further comprises

the instrumentation client controlling the compiler via the call to the interface.

8. (Currently amended)The method of claim 7, wherein the predetermined transformation call is one of a group consisting of: InsertNewBytecodesBefore; InsertNewBytecodesAfter; DeleteBytecodes; ReplaceClassfile; ReplaceMethod; InsertJNICallBefore; and InsertJNICallAfterclass.

9. (Currently amended)An information handling system comprising a processor, a mixed mode virtual machine ~~(VM)~~ and a dynamic instrumentation interface operably coupling the ~~VM~~ virtual machine and a program instrumentation tool, the ~~VM~~ virtual machine comprising:

a compiler operably coupled to the interface and responsive to signaling from the interface to transform a program, and

plural instructions comprising class loader instructions operable for determining whether a first class is loaded, and when the first class is loaded, providing first class information from a class loader to the compiler; wherein the processor is operably configured to execute said plural instructions.

10. (Currently amended) The system of claim 9, wherein the compiler comprises a just in time ~~(JIT)~~ compiler, the ~~VM~~ virtual machine ~~comprises~~ plural instructions ~~and the processor is operably configured to execute said plural instructions, the plural instructions comprising:~~

~~class loader instructions operable for determining whether a first class is loaded, and when the first class is loaded, providing first class information from a class loader to the JIT compiler~~ further comprise~~; and~~ transformation instructions operable for controlling the ~~JIT~~ just in time compiler to compile the first class into a processor executable program and to transform the executable program into a transformed program.

11. (Currently amended) The system of claim 10, wherein the ~~VM~~ virtual machine is one of a group of a Java virtual machine and a .NET virtual machine, and the class loader instructions are further operable, when the first class is not loaded, to store transformation information relating to the first class and monitor the ~~VM~~ virtual machine to determine when the first class is loaded.

12. (Original)The system of claim 11, wherein the class loader instructions for monitoring and determining when the first class is loaded are performed by a class loader responsive to first transformation signaling via the interface.

13. (Currently amended)The system of claim 10, wherein the ~~VM~~ virtual machine is one of the group of a Java virtual machine and a .NET virtual machine, the transformation instructions comprising further instructions operable for, when the first class is loaded, allowing a current instantiation of the first class to run until terminated.

14. (Currently amended)The system of claim 13, further comprising operating instructions independent of the ~~VM~~ virtual machine, wherein the transformed program is operably executed by the operating instructions independent of the ~~VM~~ virtual machine.

15. (Original)The system of claim 10, the transformation instructions comprising further instructions configured to, when the first class is already loaded, replace currently running code based on an old-object method of the first class with the transformed program and adjust an activation stack so existing invocations of the old-object method continue executing after adjusting the activation stack.

16. (Original)The system of claim 9, further comprising a client tool program having client transformation instructions operable for initiating a transformation request to the interface via a predetermined transformation call, thereby controlling the compiler via the call to the interface.

17. (Currently amended)The system of claim 16, wherein the interface comprises further instructions responsive to the predetermined call, the call being one of a group consisting of: InsertNewBytecodesBefore; InsertNewBytecodesAfter; DeleteBytecodes; ReplaceClassfile; ReplaceMethod; InsertJNICallBefore; and~~,~~ InsertJNICallAfterclass.

18. (Currently amended) A program product in a signal bearing medium, the program product comprising instructions executable by a device for presenting a hierarchical representation of a target program, the product comprising: ~~VM~~ virtual machine instructions operable as a mixed-mode virtual machine ~~(VM)~~ comprising a compiler; interface instructions operable as a dynamic

instrumentation interface for coupling the ~~VM~~ virtual machine and a program instrumentation tool, further operable responsive to signaling from the interface to transform a program being operated on by the ~~VM~~ virtual machine.

19. (Currently amended)The program product of claim 18, wherein the compiler is operable as a just in time (JIT) compiler, the ~~VM~~ virtual machine instructions further comprising: class loader instructions operable for determining whether a first class is loaded, and when the first class is loaded, providing first class information from a class loader to the JIT compiler; and transformation instructions operable for controlling the JIT compiler to compile the first class into a processor executable program and to transform the executable program into a transformed program.

20. (Currently amended) The program product of claim 19, wherein the ~~VM~~ virtual machine is operable as one of a group of a Java virtual machine and a .NET virtual machine, and the class loader instructions are further configured to, when the first class is not loaded, store transformation information relating to the first class and monitor the ~~VM~~ virtual machine to determine when the first class is loaded.

21. (Original)The program product of claim 20, wherein the class loader instructions for monitoring and determining when the first class is loaded are configured to be performed by a class loader responsive to first transformation signaling via the interface.

22. (Currently amended) The program product of claim 19, wherein the ~~VM~~ virtual machine is operable as one of the group of a Java virtual machine and a .NET virtual machine, the transformation instructions comprising further instructions configured to, when the first class is loaded, permit a current instantiation of the first class to run until terminated.

23. (Currently amended)The program product of claim 22, wherein the transformed program is

configured to be operably executed by operating instructions independent of the ~~VM~~ virtual machine.

24. (Original)The program of claim 19, the transformation instructions comprising further instructions configured to, when the first class is already loaded, replace currently running code based on an old-object method of the first class with the transformed program and adjust an activation stack so existing invocations of the old-object method continue executing after adjusting the activation stack.

25. (Original) The program product of claim 18, further comprising client transformation instructions operably part of a client tool and configured to initiate a transformation request to the interface via a predetermined transformation call, thereby controlling the compiler via the call to the interface.

26. (Original)The program product of claim 25, wherein the interface instructions are further configured to be responsive to the predetermined transformation call, the call being one of a group consisting of: InsertNewBytecodesBefore; InsertNewBytecodesAfter; DeleteBytecodes; ReplaceClassfile; ReplaceMethod; InsertJNICallBefore; and. InsertJNICallAfterclass.